10 WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION

10.1 SATISFY THE CUSTOMER AND CONTINUOUS DELIVERY OF VALUE

The Agile principle being discussed in this section is ‘Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.’ We will look at the concept of continuous delivery in more detail, and how to perceive value in a context; for example, how to order the backlog focused on RoI (Return on Investment) rather than simply reductions in cost.

The sooner a product is delivered, the sooner the product starts to produce business value and the sooner business and technical feedback can be sought. Additionally and importantly, the sooner teams can inspect a product via feedback, the sooner they can adapt to deliver the appropriate stories as they evolve.

The continuous delivery of value is why Agile aims for deliveries of valuable useable features from small delivery batch sizes (iterations/sprints). Agile deliveries can be made (and sometimes can only be made) within releases, projects etc as we have discussed elsewhere in this book; however, the longer the team waits to deliver valuable features the bigger the delivery batch size with the inherent problems that causes (risk of confusion, errors, low productivity etc).

In today’s world, it appears, there are largely two types of business:

- businesses that are ‘IT businesses’; and
- businesses that are fundamentally reliant on IT.

Therefore if the delivery of IT-driven business value is restricted in its ability to be fast and flexible, then the business that relies on that delivery will be severely, if not potentially fatally, constrained.

10.1.1 Product flow and business value

Effective product flow, i.e. ensuring that the right products are developed as continuously, is arguably the key driver to generating value in any product development, whether it is IT or otherwise. A lot of the discussion in this section is based on thinking from Donald G Reinertsen and his book on product flow: *The principles of product development flow: second generation lean product development* (Reinertsen, 2009).

In his book Reinertsen lists eight major themes that lie at the core of flow-based product development. These are:

- **Economics** – establish an integrated economically based decision-making framework; if you only understand one thing understand the cost of delay.
- **Queues** – identify work queues and endeavour to make them as short as

possible, because long queues, amongst other things, add risk and increase cycle time.

- **Variability** – within software development teams may be creating a new product, therefore variability is required. Removing variability may remove innovation. Therefore the system to be implemented must be able to deal with variability.
- **Batch size** – large batch sizes create large variability and an environment where many skills may be needed to enable delivery of a single product feature, leading to queues caused by handoffs between roles.
- **Work-in-progress constraints** – apply work-in-progress constraints to match work in progress to capacity, therefore helping to remove queues.
- **Cadence, synchronisation and flow control** – create a delivery heartbeat (cadence) within short timescales to ensure fast feedback cycles and predictability (removal of 'noise'). Synchronise across many teams on a standard, regular and short delivery cadence.
- **Fast feedback** – enables effective communication between technical teams and between technical teams and the business; this leads to more accurate products.
- **Decentralised control** – this leads to faster and more accurate decisions.

Although Agile approaches support many of Reinertsen's themes, in the following we will look at 'economics' as it is directly related to quantifying and measuring business value.

Reinertsen proposes 21 different economic principles within the 'Economics' theme. The core principle we want to discuss in this section is principle three, which looks at the 'cost of delay'. This principle is based on principles one and two:

Economic principle number one is 'the principle of quantified overall economics: select actions based on quantified overall economic impact' (Reinertsen, 2009). This principle drives the definition of value within the organisation using an overall economic framework. The intent of the framework is to manage complex multi-variable decisions by expressing all operational impacts (e.g. waste, cycle time, efficiency) in the same unit of measure – which is 'lifecycle profit impact', i.e. the impact on business profit from varying any variable.

When a business applies this principle, it enables customers and stakeholders to define an agreed ordering of the backlog based primarily on lifecycle profit impact. Therefore, lifecycle profit impact is focused on the overall value of delivery rather than just cost reduction or any other single aspect of value delivery.

Economic principle number two is the principle of interconnected dynamics, meaning that no one thing can be changed in an organisation without affecting other things. For example, the customer may want to add more stories before implementing anything. They then make the (erroneous) decision to reduce testing and refactoring to enable quick implementation of the stories. This decision may impact on the following:

- In the short term the cycle time may be decreased and therefore delivery of stories increased.
- The development cost in the short-term may decrease.
- The perceived short-term value-add may increase; however, long-term value-add may significantly decrease.
- The support and maintenance cost of the product may significantly increase as many more defects might occur later in the lifecycle when they are far more costly to fix.
- Significant technical debt may be added to the system, which will make the system very difficult and costly to enhance in the future.

The product flow solution is to create an overall interconnected project economic framework. The framework treats projects as a black box, and the focus of each project is to produce lifecycle profits. Reinertsen highlights five interconnected key measurements for the project, all of which contribute towards overall lifecycle profits. These are:

- Cycle time – work start to finish time.
- Product cost – the cost of the whole product lifecycle, from cradle to grave, i.e. the costs to manufacture the product. For pure software products arguably not that significant, although things like license costs or any other transaction costs should be included. For products that have hardware this is more significant, as it includes the costs of the parts and labour to put it together.
- Product value – the revenue the product generates over its lifetime. It is important to have this especially when considering cost of delay as delivering late can have a negative effect on revenue if a market opportunity is missed.
- Development expense – the development costs of the product.
- Risk – a quantification of the level of risk an organisation is willing to take on various parts of the development. It involves using probability of achieving a certain level of revenue balanced against what it costs to produce it.

If a team can ascertain the lifecycle profits of stories, they can then ascertain the value of each story and therefore the relative cost of delaying any story, or indeed bringing forward higher-value stories.

Economic principle number three is ‘the principle of quantified cost of delay: if you only quantify one thing, quantify the cost of delay’ (Reinertsen, 2009). The cost of delay is the cost of something being delayed on the profitability of the business.

If all variables under one single unit – defined as lifecycle profit impact – can be brought together and the interconnectivity between decisions using this unit be visualised, the cost of delay can be quantified.

Quantifying the cost of delay enables the customer to identify the relative cost of stories being available now, sooner or later. Quantifying cost of delay in lifecycle profit impact enables all collaboration and conversations about relative ordering and priority to be moved away from simplistic cost-cutting and moved towards business value conversations.

Within an Agile team the ability to exactly quantify the cost of delay is less important as the exact mathematics is not the essential aim; instead, the ability to visualise the relative cost of delay across stories within the backlog is vital to the customer being able to order the backlog effectively.

10.1.2 The product flow economic model and prioritisation

A relatively common mistake made by Agile teams is to qualify the ordering of stories in the backlog from the perspective of a single economic variable (although Reinertsen does suggest that if only one thing is measured – which it shouldn't be – it should be cost of delay).

For example, if the team is too technically focused, there is a risk that the only variable being considered is development cost, and therefore there is insufficient focus on whether (or not) value is actually being added to the business.

Another example is when the customer orders activity based only on business value and does not take into account development cost. In this situation lots of stories may be delivered, but because development cost and sensible sequencing of development resources has not been taken into account, the cost and timescale of developing the product may be significantly larger than necessary.

The sequencing of the backlog needs to take into account the overall economic model, and the related variables. Once the cost of delay of a particular story in a particular backlog has been ascertained, the customer can then think about how to sequence the backlog by using the product flow model. When considering cost of delay, the following economic approaches can be applied to ordering a backlog:

FIFO: First in First Out. The first feature that comes into the backlog is the next feature that is delivered. This may be an appropriate way to manage an incidents backlog where all the incidents are of the same importance, however, it will not be appropriate for many product developments.

SJF: Shortest Job First. If the cost of delay is equal across stories, then order the stories with the least duration first.

HCDF: High Cost of Delay First. If the cost of delay is not equal but the duration to deliver the feature is equal, then order the feature with the highest cost of delay first.

WSJF: Weighted Shortest Job First. Where nothing is equal, do the weighted shortest job first. The weighted shortest job = cost of delay/duration

10.2 DELIVER WORKING SOFTWARE FREQUENTLY

The Agile principle described in this section is 'Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.'

So what exactly is meant by 'working software'? Essentially, 'working software' is when the software is 'done' (see Section 10.2.1). The ultimate aim is that working, 'done' software is always delivered to the live environment where the customer gains benefits.

There are numerous benefits of frequently delivering working software.

- **Regular delivery cadence** Most Agile frameworks recommend that a regular delivery heartbeat or cadence of the same length is implemented. This cadence is associated with sprints/iterations (typically every 2 to 4 weeks), and/or releases, which contain multiple sprints/iterations of typically every 1 to 3 months. Setting a continuous sprint/iteration cadence enables teams to focus on delivering in a regular time frame and allows stakeholders to make themselves available at regular points to review what is being created by the team. A regular short delivery cadence enables more frequent feedback from the stakeholders; also, the team are much less likely to experience 'noise' (unplanned interruptions during iterations) that could seriously affect the team's delivery capability.
- **Commitment and trust** It is next to impossible to implement Agile successfully if teams continually miss their iteration/sprint goals. Many Agile frameworks value commitment against a clearly stated iteration/sprint goal, and an environment of trust cannot be built if the team continually fail to deliver against those commitments.

  A common cause of delivery failure is that working product is not delivered frequently enough. A team's delivery focus can be lost easily if teams cannot see an achievable goal within a timescale that they are able to commit to, or alternatively it might be that the goal as originally agreed keeps changing.
- **Understanding** A common cause of misalignment between business stakeholders and technical teams is that they do not understand each other. This can be exacerbated if delivery is being driven from detailed specification documents, as it is unlikely that both parties will interpret the document in the same way. The frequent delivery of working software ensures that the customer and stakeholders have full visibility and so understand what is being delivered.
- **Enabling 'Lean start-up'** (see Section 14.7) The most effective way to validate the market-fitness of a product is when the actual end users interact with the working product. Regular releases of new functionality allow the end user to try out the evolving product and feedback instantly.
- **Continuous improvement** Delivering frequently allows the team to regularly reflect upon their work and adapt the delivery processes as required.
- **Reduced business risk** Short feedback cycles are essential for a successful product delivery. If interactions between teams and key stakeholders are rare

it is highly likely that the wrong product will be built. When working software is delivered regularly, the business risks and associated costs are significantly reduced as this means that via inspection of the product the customer can ensure the business case is being achieved and is still robust.

There is a philosophy in Agile called 'fail fast', meaning that if the requirements of the business case are not met, the delivery can fail fast before too much time and money is wasted. Many Agile experts see this really as 'learn fast'. The team, customer and stakeholders learn fast that they are going in the wrong direction and therefore stop before deciding which is the right direction to go in, or whether to re-start at all.

- **Reduce technical risk** By regularly releasing increments to a product the change to the product between versions is smaller. This reduces the technical risk to the business and makes implementation easier. It also means that any appropriate corrective actions can be implemented effectively, frequently and with confidence.

Product delivery every few weeks may not be practical or required within a large complex integrated systems development. In this scenario it may only be feasible to deliver integrated working software produced by multiple teams within releases of multiple iterations/sprints.

10.2.1 Definition of 'done'

The 'definition of done' (DoD) refers to the point at which the product is fit for purpose and the customer will sign it off. DoDs can be related to sprint, release and project goals.

Most teams create DoD lists that define what must be signed off before a product is at 'done' status. This may be something as simple as 'code complete' and 'tests complete' though usually it is a list that will contain items such as:

- Story done (acceptance criteria met).
- Standards and guidelines done. There are certain requirements that relate to everything being produced (such as performance, scalability, auditability etc.). Therefore it makes sense to define requirements that apply to every story as standards or guidelines.
- Code done.
- Code review done.
- Unit, integration, system UA tests done.
- Deploy to environment done.
- Documentation done.
- Refactoring done.

10.3 WORKING SOFTWARE AS A MEASURE OF PROGRESS

This section relates to the Agile Manifesto principle ‘Working software is the primary measure of progress.’

In a traditional non-Agile environment milestones are typically set against the stages of whatever delivery approach is being used. For example, in a Waterfall delivery, typical milestones will be aligned to the stages in the Waterfall process: analysis, design, build, test, implement. Associated with those milestones will be Waterfall documents, for example, requirements specifications and functional specifications.

In an Agile delivery, progress is considered to have been made when the value-added product is delivered to the live environment for the customer, thereby adding value to the customer’s business. This does not necessarily mean that delivery of value-add software is the only measure of progress (others might include defect rates or the amount of technical debt and so on), but it is certainly the core measurement.

Agile audits tend to be performed against outcomes delivered by value-added software, rather than against documents or milestones.

A significant benefit of using working systems as the primary measure of progress is that it focuses the Agile team on working in an integrated, collaborative way, as the only way they can deliver working software is as a team collaborating with the customer and stakeholders. In situations where the focus is on delivering documents or milestones associated with job types (e.g. an analyst delivering analytical products within an analysis stage) the team can become disjointed and a blame culture develops.

10.4 TECHNICAL EXCELLENCE AND GOOD DESIGN

This section discusses the Agile Manifesto principle ‘Continuous attention to technical excellence and good design enhances agility.’

As discussed before, in an Agile delivery the customer prioritises stories to ensure the right product for the emergent business need is developed. At the same time it is also important for teams to verify that they are developing the product in the right way (the development process).

Quality is at the core of Agile product development (see Section 8.9), and a successful Agile delivery is dependent on Agile teams paying close attention to technical excellence and good design. If a feature of the product is identified as being of sub-standard design the team should make sure that a story is raised to specifically address the concern.

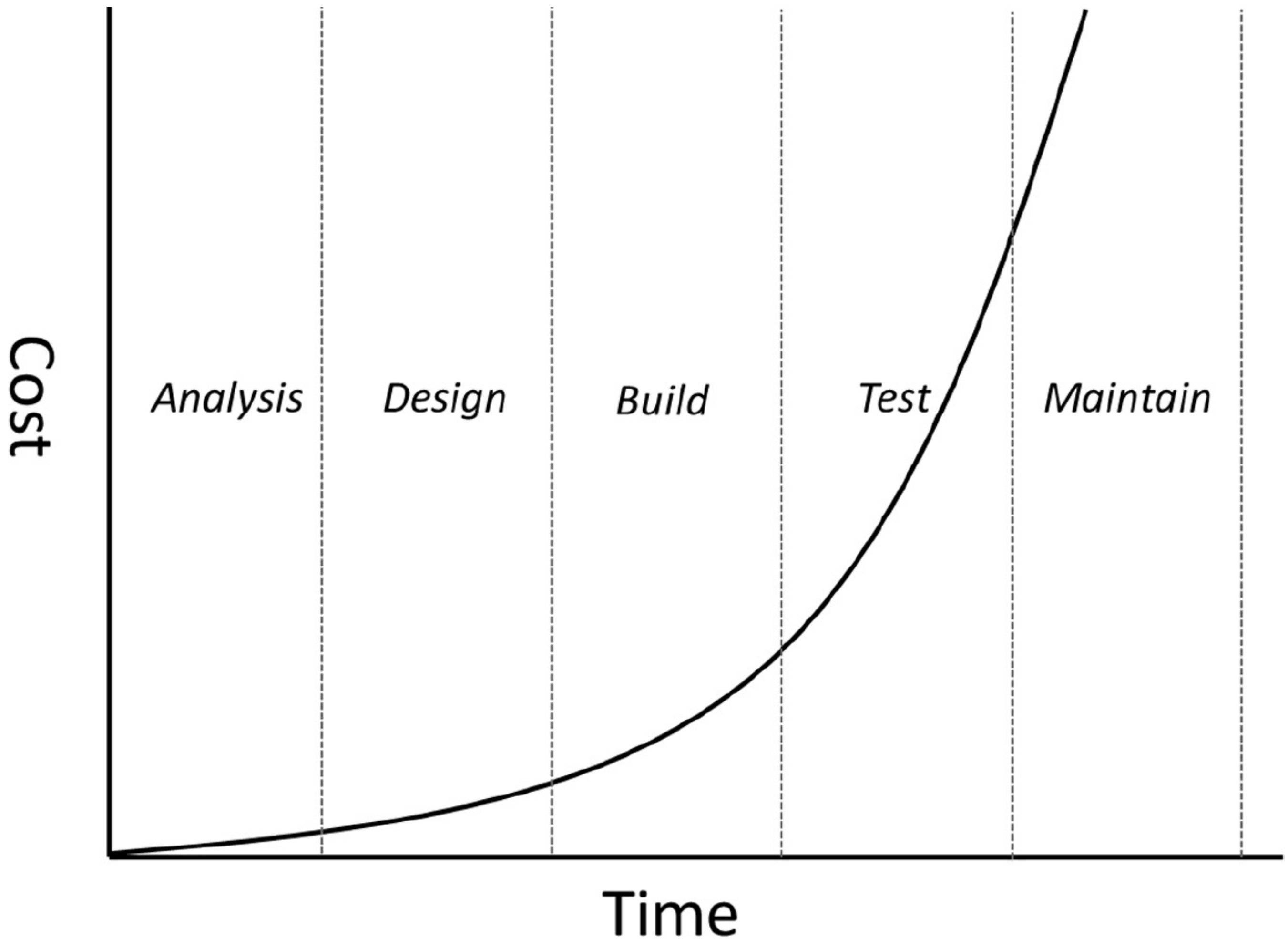
If teams do not maintain appropriate technical excellence and good design throughout the development process, it will become increasingly difficult to add further product increments as the product will get unnecessarily complex, therefore decreasing the value-add. It is too easy for the team to continue to add stories to a product without considering the overall integrated technical design of the product, and in most cases this will lead to a product that is fragile and unmaintainable.

Lack of focus on technical excellence and good design also creates great difficulties for the delivery to be Agile at all as significant time is spent dealing with the problems caused by high technical debt.

10.4.1 Technical debt

Allowing inappropriate design or architecture leads to what Agile calls 'technical debt'. Technical debt creates systems that are extremely expensive to support and maintain as they are badly designed, badly documented and typically full of defects (see Figure 10.1 - shows that technical debt pushes problems later in the delivery lifecycle where they are significantly more expensive to fix). It will also make the system very difficult to enhance, thereby restricting the ability to deliver stories frequently and add value to the business.

Figure 10.1 Relative cost for phase that a defect is fixed



Technical debt, like financial debt, accrues compound interest. For example, if teams perform a quick fix once, the next time there is a problem, they are highly likely to implement a quick fix again. This approach usually does not address underlying problems, and the effect of these problems is compounded as the product is developed further, leading to technical debt. Once technical debt has been allowed into a system it can grow very quickly and become a core design feature/problem of the system. Technical debt can be addressed through refactoring (see Section 8.10.1).

10.4.2 Architecture and design frameworks

Agile does not mandate the usage of any specific design or architecture framework, and it is outside the boundaries of this book to discuss these in detail. Agile teams may (or may not) get benefit from using architecture principles such as SOA (Service Orientated Architectures), architecture approaches such as TOGAF (The Open Group Architecture Framework) or design approaches such as UML (Unified Modelling Language).

However, when implementing any architecture or design framework it is essential to also consider another core Agile principle: Simplicity, the art of maximising the amount of work not done (see Chapter 13). It can be very easy for a team to deliver everything defined within a particular architecture or design framework without thinking specifically about which artefacts from the framework demonstrably add value. It is the responsibility of the Agile lead to coach the team to think about simplicity and to only use those aspects of a framework that demonstrably add value.